

# quanteda

October 21, 2014

**Type** Package

**Title** Quantitative Analysis of Textual Data

**Version** 0.5.7.001

**Date** 2014-10-19

**Author** Ken Benoit <kbenoit@lse.ac.uk> and Paul Nulty <p.nulty@lse.ac.uk>

**Maintainer** Ken Benoit <kbenoit@lse.ac.uk>

**Description** A package for the management and quantitative analysis of textual data with R. quanteda makes it easy to manage texts in the form of a corpus, defined as a collection of texts that includes document-level variables specific to each text, as well as meta-data for documents and for the collection as a whole. quanteda includes tools to make it easy and fast to manipulate the texts in a corpus, for instance by tokenizing them, with or without stopwords or stemming, or to segment them by sentence or paragraph units. quanteda implements bootstrapping methods for texts that makes it easy to resample texts from pre-defined units, to facilitate computation of confidence intervals on textual statistics using techniques of non-parametric bootstrapping, but applied to the original texts as data. quanteda includes a suite of sophisticated tools to extract features of the texts into a quantitative matrix, where these features can be defined according to a dictionary or thesaurus, including the declaration of collocations to be treated as single features. Once converted into a quantitative matrix (known as a ``dfm" for document-feature matrix), the textual features can be analyzed using quantitative methods for describing, comparing, or scaling texts, or topic modelling, or used to train machine learning methods for class prediction.

**License** GPL-3

**Depends** R (>= 3.0), data.table (>= 1.9.2)

**Imports** SnowballC, wordcloud, slam, tm, proxy

**Suggests**

quantedaData, austin, entropy, jsonlite, openNLP, RJSONIO, RCurl, twitteR, XML, lda, topicmodels, tcltk2, knitr

**URL** <http://github.com/kbenoit/quanteda>

**BugReports** <https://github.com/kbenoit/quanteda/issues>

**LazyData** TRUE

**VignetteBuilder** knitr

## R topics documented:

bigrams . . . . .	3
changeunits . . . . .	4
clean . . . . .	4
collocations . . . . .	5
compoundWords . . . . .	7
corpus . . . . .	8
countSyllables . . . . .	11
describeTexts . . . . .	11
dfm . . . . .	12
dfm2ldaformat . . . . .	14
dfm2tmformat . . . . .	15
directory . . . . .	15
docnames . . . . .	16
docvars . . . . .	17
encoding . . . . .	18
features.dfm . . . . .	18
flatten.dictionary . . . . .	19
getRootFileNames . . . . .	20
getTextDir . . . . .	20
getTextDirGui . . . . .	21
getTextFiles . . . . .	22
getTweets . . . . .	22
inaugCorpus . . . . .	23
kwic . . . . .	24
language . . . . .	25
metacorporus . . . . .	25
metadoc . . . . .	26
ndoc . . . . .	27
ngrams . . . . .	27
nresample . . . . .	28
plot.dfm . . . . .	29
print.dfm . . . . .	30
quanteda . . . . .	30
readLIWCdict . . . . .	30
readWStatDict . . . . .	31
resample . . . . .	32
segmentSentence . . . . .	33
settings . . . . .	34
settingsInitialize . . . . .	35
similarity . . . . .	36
sort.dfm . . . . .	37
statLexdiv . . . . .	38
stopwords . . . . .	40
stopwordsGet . . . . .	41
stopwordsRemove . . . . .	41
subset.corpus . . . . .	42

summary.corpus . . . . .	43
syllableCounts . . . . .	43
texts . . . . .	44
tf . . . . .	44
tfidf . . . . .	45
tokenize . . . . .	46
topfeatures . . . . .	47
trimdfm . . . . .	47
uk2010immig . . . . .	48
wordstem . . . . .	49
zipfiles . . . . .	49

<b>Index</b>	<b>51</b>
--------------	-----------

---

bigrams	<i>Create bigrams</i>
---------	-----------------------

---

**Description**

Create bigrams

**Usage**

```
bigrams(text, window = 1, concatenator = "_", include.unigrams = FALSE,
...)
```

**Arguments**

- text            character vector containing the texts from which bigrams will be constructed
- window        how many words to be counted for adjacency. Default is 1 for only immediately neighbouring words. This is only available for bigrams, not for ngrams.
- concatenator   character for combining words, default is \_ (underscore) character
- include.unigrams        if TRUE, return unigrams as well
- ...            provides additional arguments passed to [tokenize](#)

**Value**

a character vector of bigrams

**Author(s)**

Ken Benoit and Kohei Watanabe

**Examples**

```
bigrams("The quick brown fox jumped over the lazy dog.")
bigrams(c("The quick brown fox", "jumped over the lazy dog."))
bigrams(c("The quick brown fox", "jumped over the lazy dog."), window=2)
```

---

changeunits

*change the document units of a corpus*


---

### Description

For a corpus, recast the documents down or up a level of aggregation. "Down" would mean going from documents to sentences, for instance. "Up" means from sentences back to documents. This makes it easy to reshape a corpus from a a collection of documents into a collection of sentences, for instance.

### Usage

```
changeunits(corp, to = c("sentences", "paragraphs", "documents"), ...)
```

### Arguments

corp	corpus whose document units will be reshaped
to	new documents units for the corpus to be recast in
...	passes additional arguments to <a href="#">segment</a>

### Examples

```
# simple example
mycorpus <- corpus(c(textone="This is a sentence. Another sentence. Yet another.",
                      texttwo="Premiere phrase. Deuxieme phrase."),
                  docvars=list(country=c("UK", "USA"), year=c(1990, 2000)),
                  notes="This is a simple example to show how changeunits() works.")
language(mycorpus) <- c("english", "french")
summary(mycorpus)
summary(changeunits(mycorpus, to="sentences"), showmeta=TRUE)

# example with inaugural corpus speeches
mycorpus2 <- subset(inaugCorpus, Year>2004)
mycorpus2
paragCorpus <- changeunits(mycorpus2, to="paragraphs")
paragCorpus
summary(paragCorpus, 100, showmeta=TRUE)
## Note that Bush 2005 is recorded as a single paragraph because that text used a single
## \n to mark the end of a paragraph.
```

---

clean

*simple cleaning of text before processing*


---

### Description

clean removes punctuation and digits from text, using the regex character classes for punctuation and digits. clean uses the standard R function tolower to convert the text to lower case. Each of these steps is optional, but switched on by default, so for example, to remove punctuation and convert to lower, but keep digits, the command would be: clean(mytexts, removeDigits=FALSE)

**Usage**

```
clean(x, ...)

## S3 method for class 'character'
clean(x, removeDigits = TRUE, removePunct = TRUE,
      lower = TRUE, additional = NULL, twitter = TRUE, ...)

## S3 method for class 'corpus'
clean(x, removeDigits = TRUE, removePunct = TRUE,
      lower = TRUE, additional = NULL, twitter = TRUE, ...)
```

**Arguments**

<code>x</code>	The object to be cleaned. Can be either a character vector or a corpus object. If <code>x</code> is a corpus, <code>clean</code> returns a copy of the <code>x</code> with the texts cleaned.
<code>...</code>	additional parameters
<code>removeDigits</code>	remove numbers if TRUE
<code>removePunct</code>	remove punctuation if TRUE
<code>lower</code>	convert text to lower case TRUE
<code>additional</code>	additional characters to remove ( <a href="#">regular expression</a> )
<code>twitter</code>	if TRUE, do not remove @ or #

**Value**

A character vector equal in length to the original texts, after cleaning.

**Examples**

```
clean("This is 1 sentence with 2.0 numbers in it, and one comma.", removeDigits=FALSE)
clean("This is 1 sentence with 2.0 numbers in it, and one comma.", lower=FALSE)
clean("We are his Beliebers, and him is #ourjustin @justinbieber we love u", twitter=TRUE)
clean("Collocations can be represented as inheritance_tax using the _ character.")
clean("But under_scores can be removed using the additional argument.", additional="[_]")

# for a vector of texts
clean(c("This is 1 sentence with 2.0 numbers in it, and one comma.",
        "$1.2 billion was spent on text analysis in 2014."))
```

---

collocations

---

*Detect collocations from text*


---

**Description**

Detects collocations (currently, bigrams) from texts or a corpus, returning a data.frame of collocations and their scores, sorted by the likelihood ratio  $G^2$  and Pearson's  $\chi^2$ .

**Usage**

```
collocations(x, ...)

## S3 method for class 'character'
collocations(x, method = c("lr", "chi2", "pmi", "dice",
  "all"), n = 2, top = NULL, ...)

## S3 method for class 'corpus'
collocations(x, method = c("lr", "chi2", "pmi", "dice",
  "all"), n = 2, top = NULL, ...)
```

**Arguments**

x	a text, a character vector of texts, or a corpus
...	additional parameters
method	association measure for detecting collocations. Available measures for bigrams are: "lr" The likelihood ratio statistic $G^2$ , computed as: $2 * \sum_i \sum_j (n_{ij} * \log \frac{n_{ij}}{m_{ij}})$ "chi2" Pearson's $\chi^2$ statistic, computed as: $\sum_i \sum_j \frac{(n_{ij} - m_{ij})^2}{m_{ij}}$ "pmi" point-wise mutual information score, computed as $\log n_{11}/m_{11}$ "dice" the Dice coefficient, computed as $n_{11}/n_{1.} + n_{.1}$ "all" returns all of the above
n	length of the collocation. Only bigrams (n=2) implemented so far.
top	the number of collocations to return, sorted in descending order of the requested statistic, or $G^2$ if none is specified.

**Value**

A data.frame of collocations, their frequencies, and the computed association measure.

**Author(s)**

Kenneth Benoit

**References**

Add some.

**See Also**

bigrams, trigrams

**Examples**

```
collocations(inaugTexts, top=10)
collocations(inaugCorpus, top=10, method="chi2")
```

---

compoundWords	<i>convert phrases into single tokens</i>
---------------	---

---

## Description

Replace multi-word phrases in text(s) with a compound version of the phrases concatenated with connector (by default, the "\_" character) to form a single token. This prevents tokenization of the phrases during subsequent processing by eliminating the whitespace delimiter.

## Usage

```
compoundWords(txts, dictionary, connector = "_")
```

## Arguments

txts	character or character vector of texts
dictionary	a list or named list (such as a quanteda dictionary) that contains some phrases, defined as multiple words delimited by whitespace. These can be up to 9 words long.
connector	the concatenation character that will connect the words making up the multi-word phrases. The default _ is highly recommended since it will not be removed during normal cleaning and tokenization (while nearly all other punctuation characters, at least those in the POSIX class <code>[[:punct:]]</code> ) will be removed.

## Value

character or character vector of texts with phrases replaced by compound "words" joined by the connector

## Examples

```
mytexts <- c("The new law included a capital gains tax, and an inheritance tax.",
            "New York City has raised a taxes: an income tax and a sales tax.")
mydict <- list(tax=c("tax", "income tax", "capital gains tax", "inheritance tax"))
(cw <- compoundWords(mytexts, mydict))
print(dfm(cw), show.values=TRUE)

# when used as a dictionary for dfm creation
mydfm2 <- dfm(cw, dictionary=lapply(mydict, function(x) gsub(" ", "_", x)))
print(mydfm2, show.values=TRUE)
# to pick up "taxes" in the second text, set regular_expression=TRUE
mydfm3 <- dfm(cw, dictionary=lapply(mydict, function(x) gsub(" ", "_", x)),
              dictionary_regex=TRUE)
print(mydfm3, show.values=TRUE)
```

corpus

*Constructor for corpus objects***Description**

Creates a corpus from a document source. The current available document sources are:

- a character vector (as in R class `char`) of texts;
- a directory of text files, using [directory](#);
- a directory constructed from a zip file consisting of text files, using [zipfiles](#); and
- a **tm** [VCorpus](#) class corpus object, meaning that anything you can use to create a **tm** corpus, including all of the tm plugins plus the built-in functions of tm for importing pdf, Word, and XML documents, can be used to create a quanteda [corpus](#).

Corpus-level meta-data can be specified at creation, containing (for example) citation information and notes, as can document-level variables and document-level meta-data.

**Usage**

```
corpus(x, ...)
```

```
## S3 method for class 'directory'
corpus(x, enc = NULL, docnames = NULL,
       docvarsfrom = c("none", "filenames", "headers"), docvarnames = NULL,
       sep = "_", pattern = "\\..txt$", source = NULL, notes = NULL,
       citation = NULL, ...)
```

```
## S3 method for class 'twitter'
corpus(x, enc = NULL, notes = NULL, citation = NULL,
       ...)
```

```
## S3 method for class 'VCorpus'
corpus(x, enc = NULL, notes = NULL, citation = NULL,
       ...)
```

```
## S3 method for class 'character'
corpus(x, enc = NULL, docnames = NULL, docvars = NULL,
       source = NULL, notes = NULL, citation = NULL, ...)
```

```
is.corpus(x)
```

```
## S3 method for class 'corpus'
c1 + c2
```

**Arguments**

<code>x</code>	A source of texts to form the documents in the corpus. This can be a filepath to a directory containing text documents (see <a href="#">directory</a> ), or a character vector of texts.
<code>...</code>	additional arguments



enc	A string specifying the input encoding for texts in the corpus. Must be a valid entry in <code>iconvlist()</code> , since the code in <code>corpus.character</code> will convert this to UTF-8 using <code>iconv</code> . Currently only one input encoding can be specified for a collection of input texts, meaning that you should not mix input text encoding types in a single corpus call.
docnames	Names to be assigned to the texts, defaults to the names of the character vector (if any), otherwise assigns "text1", "text2", etc.
docvarsfrom	Argument to specify where docvars are to be taken, from parsing the filenames separated by <code>sep</code> or from meta-data embedded in the text file header (headers).
docvarnames	Character vector of variable names for docvars
sep	Separator if <code>docvars</code> names are taken from the filenames.
pattern	filename extension - set to "*" if all files are desired. This is a <a href="#">regular expression</a> .
source	A string specifying the source of the texts, used for referencing.
notes	A string containing notes about who created the text, warnings, To Dos, etc.
citation	Information on how to cite the corpus.
docvars	A data frame of attributes that is associated with each text.
c1	corpus one to be added
c2	corpus two to be added

## Details

The `+` operator for a corpus object will combine two corpus objects, resolving any non-matching `docvars` or `metadoc` fields by making them into NA values for the corpus lacking that field. Corpus-level meta data is concatenated, except for source and notes, which are stamped with information pertaining to the creation of the new joined corpus.

There are some issues that need to be addressed in future revisions of `quanteda` concerning the use of factors to store document variables and meta-data. Currently most or all of these are not recorded as factors, because we use `stringsAsFactors=FALSE` in the `data.frame` calls that are used to create and store the document-level information, because the texts should always be stored as character vectors and never as factors.

## Value

A corpus class object containing the original texts, document-level variables, document-level meta-data, corpus-level metadata, and default settings for subsequent processing of the corpus. A corpus consists of a list of elements described below, although these should only be accessed through accessor and replacement functions, not directly (since the internals may be subject to change). The structure of a corpus classed list object is:

<code>\$documents</code>	A data frame containing the document level information, consisting of <code>texts</code> , user-named <code>docvars</code> variables describing attributes of the documents, and <code>metadoc</code> document-level metadata whose names begin with an underscore character, such as <code>_language</code> .
<code>\$metadata</code>	A named list set of corpus-level meta-data, including source and created (both generated automatically unless assigned), notes, and citation.
<code>\$settings</code>	Settings for the corpus which record options that govern the subsequent processing of the corpus when it is converted into a document-feature matrix ( <code>dfm</code> ). See <a href="#">settings</a> .

`$tokens` An indexed list of tokens and types tabulated by document, including information on positions. Not yet fully implemented.

`is.corpus` returns TRUE if the object is a corpus

### Note

When `x` is a [VCorpus](#) object, the fixed metadata fields from that object are imported as document-level metadata. Currently no corpus-level metadata is imported, but we will add that soon.

### See Also

[docvars](#), [metadoc](#), [metacorporus](#), [language](#), [encoding](#), [settings](#), [texts](#)

### Examples

```
## Not run:
# import texts from a directory of files
summary(corpus(directory("~/Dropbox/QUANTESS/corpora/ukManRenamed"),
  enc="UTF-8", docvarsfrom="filenames",
  source="Ken's UK manifesto archive",
  docvarnames=c("Country", "Level", "Year", "language")), 5))
summary(corpus(directory("~/Dropbox/QUANTESS/corpora/ukManRenamed"),
  enc="UTF-8", docvarsfrom="filenames",
  source="Ken's UK manifesto archive",
  docvarnames=c("Country", "Level", "Year", "language", "Party")), 5))

# choose a directory using a GUI
corpus(directory())

# from a zip file on the web
myzipcorp <- corpus(zipfiles("http://kenbenoit.net/files/EUcoalsubsidies.zip"),
  notes="From some EP debate about coal mine subsidies")
docvars(myzipcorp, speakernames=docnames(myzipcorp))
summary(myzipcorp)

## End(Not run)
#
## import a tm VCorpus
if (require(tm)) {
  data(crude) # load in a tm example VCorpus
  mytmCorpus <- corpus(crude)
  summary(mytmCorpus, showmeta=TRUE)
}
#
# create a corpus from texts
corpus(inaugTexts)

# create a corpus from texts and assign meta-data and document variables
uk2010immigCorpus <- corpus(uk2010immig,
  docvars=data.frame(party=names(uk2010immig)),
  enc="UTF-8")
```

---

countSyllables	Returns a count of the number of syllables in the input
----------------	---

---

### Description

This function takes a text and returns a count of the number of syllables it contains. For British English words, the syllable count is exact and looked up from the CMU pronunciation dictionary. For any word not in the dictionary the syllable count is estimated by counting vowel clusters.

### Usage

```
countSyllables(sourceText)
```

### Arguments

sourceText	Character vector of texts whose syllables will be counted
------------	---

### Details

This only works for English.

### Value

numeric Named vector of counts of the number of syllables for each element of sourceText. When a word is not available in the lookup table, its syllables are estimated by counting the number of (English) vowels in the word.

### Examples

```
countSyllables("This is an example sentence.")
myTexts <- c("Text one.", "Superduper text number two.", "One more for the road.")
names(myTexts) <- paste("myText", 1:3, sep="")
countSyllables(myTexts)
```

---

describeTexts	print a summary of texts
---------------	--------------------------

---

### Description

Prints to the console a description of the texts, including number of types, tokens, and sentences

### Usage

```
describeTexts(txts, verbose = TRUE)
```

### Arguments

txts	The texts to be described
verbose	Default is TRUE. Set to false to suppress output messages

## Examples

```
describeTexts(c("testing this text", "and this one"))
describeTexts(uk2010immig)
```

dfm

*Create a document-feature matrix from a corpus object*

## Description

Returns a document by feature matrix with additional meta-information (settings, identification of training texts for supervised models, resampling information, etc.) that is useful in other quanteda functions. A typical usage would be to produce a word-frequency matrix where the cells are counts of words by document, but the definition of "features" is entirely general.

## Usage

```
dfm(x, ...)

## S3 method for class 'corpus'
dfm(x, feature = c("word"), stem = FALSE,
     stopwords = NULL, bigram = FALSE, groups = NULL, verbose = TRUE,
     dictionary = NULL, thesaurus = NULL, dictionary_regex = FALSE,
     keep = NULL, bootstrap = FALSE, addto = NULL, ...)

## S3 method for class 'character'
dfm(x, feature = c("word"), stem = FALSE,
     stopwords = NULL, bigram = FALSE, verbose = TRUE, dictionary = NULL,
     thesaurus = NULL, dictionary_regex = FALSE, keep = NULL, addto = NULL,
     ...)

is.dfm(x)

as.dfm(x)
```

## Arguments

x	Corpus or character vector from which to generate the document-feature matrix
...	additional arguments passed to <a href="#">clean</a>
feature	Feature to count (e.g. words)
stem	Stem the words
stopwords	A character vector of stopwords that will be removed from the text when constructing the <a href="#">dfm</a> . If NULL (default) then no stopwords will be applied. If "TRUE" then it currently defaults to <a href="#">stopwords</a> .
bigram	include bigrams as well as unigram features, if TRUE
groups	Grouping variable for aggregating documents
verbose	Get info to screen on the progress
dictionary	A list of character vector dictionary entries, including regular expressions (see examples)

thesaurus	A list of character vector "thesaurus" entries, in a dictionary list format, which can also include regular expressions if <code>dictionary_regex</code> is TRUE (see examples). Note that unlike dictionaries, each entry in a thesaurus key must be unique, otherwise only the first match in the list will be used. Thesaurus keys are converted to upper case to create a feature label in the dfm, as a reminder that this was not a type found in the text, but rather the label of a thesaurus key.
dictionary_regex	TRUE means the dictionary is already in regular expression format, otherwise it will be converted from "wildcard" format
keep	a regular expression specifying which features to keep
bootstrap	if TRUE, compute multiple dfm's from resampled texts in the corpus. Requires a resampled corpus. See <a href="#">resample</a> .
addto	NULL by default, but if an existing dfm object is specified, then the new dfm will be added to the one named. If both <a href="#">dfm</a> 's are built from dictionaries, the combined dfm will have its <code>Non_Dictionary</code> total adjusted.

## Details

`is.dfm` returns TRUE if and only if its argument is a [dfm](#).

`as.dfm` coerces a matrix to a dfm

## Value

A specially classed matrix object with row names equal to the document names and column names equal to the feature labels. Additional information is attached to this object as [attributes](#), such as [settings](#).

## Author(s)

Kenneth Benoit

## Examples

```
wfm <- dfm(inaugCorpus)

## by president, after 1960
wfmByPresfrom1900 <- dfm(subset(inaugCorpus, Year>1900), groups="President")
docnames(wfmByPresfrom1900)

## with dictionaries
mycorpus <- subset(inaugCorpus, Year>1900)
mydict <- list(christmas=c("Christmas", "Santa", "holiday"),
               opposition=c("Opposition", "reject", "notincorpus"),
               taxing="taxing",
               taxation="taxation",
               taxregex="tax*",
               country="united states")
dictDfm <- dfm(mycorpus, dictionary=mydict)
print(dictDfm, show.values=TRUE)

## with the thesaurus feature
mytexts <- c("The new law included a capital gains tax, and an inheritance tax.",
             "New York City has raised a taxes: an income tax and a sales tax.")
mydict <- list(tax=c("tax", "income tax", "capital gains tax", "inheritance tax"))
```

```

print(dfm(compoundWords(mytexts, mydict),
          thesaurus=lapply(mydict, function(x) gsub("\\s", "_", x))),
      show.values=TRUE)
# pick up "taxes" with "tax" as a regex
print(dfm(compoundWords(mytexts, mydict), thesaurus=list(anytax="tax"), dictionary_regex=TRUE), TRUE)

## removing stopwords
testText <- "The quick brown fox named Seamus jumps over the lazy dog also named Seamus, with
            the newspaper from a a boy named Seamus, in his mouth."
testCorpus <- corpus(testText)
settings(testCorpus, "stopwords")
print(dfm(testCorpus, stopwords=TRUE), TRUE)

## keep only certain words
print(dfm(testCorpus, keep="s$"), TRUE) # keep only words ending in "s"
testTweets <- c("My homie @justinbieber #justinbieber getting his shopping on in #LA yesterday #beliebers",
               "To all the haters including my brother #justinbieber #justinbiebermeetcrystalalley #emabiggest",
               "Justin Bieber #justinbieber #belieber #kidrauhl #fetusjustin #EMABiggestFansJustinBieber")
print(dfm(testTweets, keep="^#"), TRUE) # keep only hashtags

```

---

dfm2ldaformat

---

Convert a *dfm* into the format needed by **lda**


---

## Description

Convert a quanteda *dfm* object into the indexed format required by the topic modelling package **lda**.

## Usage

```
dfm2ldaformat(d)
```

## Arguments

**d**                      A *dfm* object

## Value

A list with components "documents" and "vocab" as needed by [lda.collapsed.gibbs.sampler](#)

## Examples

```

mycorpus <- subset(inaugCorpus, Year>1970)
d <- dfm(mycorpus, stopwords=TRUE)
d <- trimdfm(d, minCount=5, minDoc=3)
td <- dfm2ldaformat(d)
if (require(lda)) {
  tmodel.lda <- lda.collapsed.gibbs.sampler(documents=td$documents,
                                           K=10,
                                           vocab=td$vocab,
                                           num.iterations=50, alpha=0.1, eta=0.1)
  top.topic.words(tmodel.lda$topics, 10, by.score=TRUE) # top five words in each topic
}

```

---

dfm2tmformat	Convert a <i>dfm</i> into a <b>tm</b> <i>DocumentTermMatrix</i>
--------------	---

---

### Description

**tm** represents sparse document-feature matrixes in the [simple triplet matrix](#) format of the package **slam**. This function converts a *dfm* into a *DocumentTermMatrix*, enabling a *dfm* to be used with other packages that expect this format, such as **topicmodels**.

### Usage

```
dfm2tmformat(d, weighting = weightTf)
```

### Arguments

<i>d</i>	A <i>dfm</i> object
<i>weighting</i>	weight function arguments passed to <code>as.TermDocumentMatrix</code> , defaults to term frequency (see <a href="#">as.DocumentTermMatrix</a> for a list of options, such as <code>tf-idf</code> ).

### Value

A simple triplet matrix of class [as.DocumentTermMatrix](#)

### Examples

```
mycorpus <- subset(inaugCorpus, Year>1970)
d <- trimdfm(dfm(mycorpus), minCount=5, minDoc=3)
dim(d)
td <- dfm2tmformat(d)
length(td$v)
if (require(topicmodels)) (tmodel.lda <- LDA(td, control = list(alpha = 0.1), k = 5))
```

---

directory	Function to declare a connection to a directory (containing files)
-----------	--

---

### Description

Function to declare a connection to a directory, although unlike [file](#) it does not require closing. If the directory does not exist, the function will return an error.

### Usage

```
directory(path = NULL)
```

### Arguments

<i>path</i>	String describing the full path of the directory or <code>NULL</code> to use a GUI to choose a directory from disk
-------------	--

## Examples

```
## Not run:
# name a directory of files
mydir <- directory("~/Dropbox/QUANTESS/corpora/ukManRenamed")
corpus(mydir)

# choose a directory using a GUI
corpus(directory())
## End(Not run)
```

---

docnames	<i>get or set document names</i>
----------	----------------------------------

---

## Description

Extract the document names from a corpus or a document-feature matrix. Document names are the rownames of the documents data.frame in a corpus, or the rownames of the [dfm](#) object for a dfm. of the [dfm](#) object.

docnames queries the document names of a corpus or a dfm

docnames <- assigns new values to the document names of a corpus. (Does not work for dfm objects, whose document names are fixed.)

## Usage

```
docnames(x)

## S3 method for class 'corpus'
docnames(x)

docnames(x) <- value

## S3 method for class 'dfm'
docnames(x)
```

## Arguments

x	the object with docnames
value	a character vector of the same length as x

## Value

docnames returns a character vector of the document names

docnames<- assigns a character vector of the document names in a corpus



## Examples

```
# query the document names of the inaugural speech corpus
docnames(inaugCorpus) <- paste("Speech", 1:ndoc(inaugCorpus), sep="")

# reassign the document names of the inaugural speech corpus
docnames(inaugCorpus) <- paste("Speech", 1:ndoc(inaugCorpus), sep="")
#
# query the document names of a dfm
docnames(dfm(inaugTexts[1:5]))
```

---

docvars

*get or set for document-level variables*

---

## Description

Get or set variables for the documents in a corpus

## Usage

```
docvars(x, field = NULL)

docvars(x, field = NULL) <- value
```

## Arguments

x	corpus whose document-level variables will be read or set
field	string containing the document-level variable name
value	the new values of the document-level variable

## Value

docvars returns a data.frame of the document-level variables

docvars<- assigns value to the named field

## Examples

```
head(docvars(inaugCorpus))
docvars(inaugCorpus, "President") <- paste("prez", 1:ndoc(inaugCorpus), sep="")
head(docvars(inaugCorpus))
```

---

encoding	<i>get the encoding of documents in a corpus</i>
----------	--

---

### Description

Get or set the `_encoding` document-level metadata field in a corpus.

### Usage

```
encoding(x)

encoding(x) <- value
```

### Arguments

<code>x</code>	a corpus object
<code>value</code>	a character vector or scalar representing the new value of the encoding (see Note)

### Details

This function modifies the `_encoding` value set by [metadoc](#). It is a wrapper for `metadoc(corp, "encoding")`.

### Note

This function differs from R's built-in [Encoding](#) function, which only allows the four values of "latin1", "UTF-8", "bytes", and "unknown" (and which assigns "unknown" to any text that contains only ASCII characters). Legal values for encodings must be from [iconvlist](#). Note that encoding does not convert or set encodings, it simply records a user declaration of a valid encoding. (We hope to implement checking and conversion later.)

---

features.dfm	<i>extract the feature labels from a <a href="#">dfm</a></i>
--------------	--

---

### Description

Extract the features from a document-feature matrix, which are stored as the column names of the [dfm](#) object.

### Usage

```
## S3 method for class 'dfm'
features(x)
```

### Arguments

<code>x</code>	the object (dfm) whose features will be extracted
----------------	---

**Value**

Character vector of the features

**Examples**

```
features(dfm(inaugTexts))[1:50] # first 50 features (alphabetically sorted)
```

---

flatten.dictionary	<i>Flatten a hierarchical dictionary into a list of character vectors</i>
--------------------	---

---

**Description**

Converts a hierarchical dictionary (a named list of named lists, ending in character vectors at the lowest level) into a flat list of character vectors. Works like `unlist(dictionary, recursive=TRUE)` except that the recursion does not go to the bottom level.

**Usage**

```
flatten.dictionary(elms, parent = "", dict = list())
```

**Arguments**

elms	list to be flattened
parent	parent list name, gets built up through recursion in the same way that <code>unlist(dictionary, recursive=TRUE)</code> works
dict	the bottom list of dictionary entries ("synonyms") passed up from recursive calls

**Details**

Called by `dfm()`

**Value**

A dictionary flattened down one level further than the one passed

**Author(s)**

Kohei Watanabe

**Examples**

```
dictPopulismEN <-
  list(populism=c("elit*", "consensus*", "undemocratic*", "referend*",
    "corrupt*", "propagand", "politici*", "*deceit*",
    "*deceiv*", "*betray*", "shame*", "scandal*", "truth*",
    "dishonest*", "establish*", "ruling*"))
flatten.dictionary(dictPopulismEN)

hdict <- list(level1a = list(level1a1 = c("l1a11", "l1a12"),
  level1a2 = c("l1a21", "l1a22")),
  level1b = list(level1b1 = c("l1b11", "l1b12"),
    level1b2 = c("l1b21", "l1b22", "l1b23")))
```

```

level1c = list(level1c1a = list(level1c1a1 = c("lowest1", "lowest2")),
               level1c1b = list(level1c1b1 = c("lowestalone"))))
flatten.dictionary(hdict)

```

---

getRootFileNames	<i>Truncate absolute filepaths to root filenames</i>
------------------	--

---

### Description

This function takes an absolute filepath and returns just the document name

### Usage

```
getRootFileNames(longFilenames)
```

### Arguments

longFilenames    Absolute filenames including a full path with directory

### Value

character vector of filenames withouth directory path

### Author(s)

Paul Nulty

### Examples

```

## Not run:
getRootFileNames('/home/paul/documents/libdem09.txt')

## End(Not run)

```

---

getTextDir	<i>loads all text files from a given directory</i>
------------	--

---

### Description

given a directory name, get a list of all files in that directory and load them into a character vector using getTextFiles

### Usage

```
getTextDir(dirname, pattern = "\\..txt$", enc = "unknown")
```

### Arguments

dirname	A directory path
pattern	a <a href="#">regular expression</a> pattern match for the input file names
enc	a value for encoding that is a legal value for <a href="#">Encoding</a>

**Value**

character vector of texts read from disk

**Author(s)**

Paul Nulty

**Examples**

```
## Not run:  
getTextDir('/home/paul/documents/')  
  
## End(Not run)
```

---

getTextDirGui	<i>provides a gui interface to choose a gui to load texts from</i>
---------------	--

---

**Description**

launches a GUI to allow the user to choose a directory from which to load all files.

**Usage**

```
getTextDirGui()
```

**Value**

character vector of texts read from disk

**Author(s)**

Paul Nulty

**Examples**

```
## Not run:  
getTextFiles('/home/paul/documents/libdem09.txt')  
  
## End(Not run)
```

---

getTextFiles	<i>load text files from disk into a vector of character vectors points to files, reads them into a character vector of the texts with optional names, default being filenames returns a named vector of complete, unedited texts</i>
--------------	--

---

### Description

load text files from disk into a vector of character vectors points to files, reads them into a character vector of the texts with optional names, default being filenames returns a named vector of complete, unedited texts

### Usage

```
getTextFiles(filenamees, textnames = NULL, enc = "unknown",
  verbose = FALSE)
```

### Arguments

filenamees	a vector of paths to text files
textnames	names to assign to the texts
enc	a value for encoding that is a legal value for <a href="#">Encoding</a>
verbose	If TRUE, print out names of files being read. Default is FALSE

### Value

character vector of texts read from disk

### Author(s)

Paul Nulty

### Examples

```
## Not run:
getTextFiles('/home/paul/documents/libdem09.txt')

## End(Not run)
```

---

getTweets	<i>Function to declare a twitter search</i>
-----------	---

---

### Description

Function to declare a connection to a twitter search

### Usage

```
getTweets(query, numResults = 50, key, cons_secret, token, access_secret)
```

**Arguments**

query	String describing the search query terms
numResults	Number of tweets to return. Maximum of approximately 1500
key	Key for twitter API authentication
cons_secret	for twitter API authentication
token	String for twitter API authentication
access_secret	for twitter API authentication

**Value**

The search results marked as a 'twitter' object for use by corpus.twitter()

---

inaugCorpus	<i>A corpus of US presidential inaugural addresses from 1789-2013</i>
-------------	---

---

**Description**

inaugCorpus is the [quanteda](#) corpus object of US presidents' inaugural addresses since 1789. Document variables contain the year of the address and the last name of the president.

inaugTexts is the character vector of US presidential inauguration speeches

**References**

<https://archive.org/details/Inaugural-Address-Corpus-1789-2009> and <http://www.presidency.ucsb.edu/inaugurals.php>.

**Examples**

```
# some operations on the inaugural corpus
data(inaugCorpus)
summary(inaugCorpus)
head(docvars(inaugCorpus), 10)
# working with the character vector only
data(inaugTexts)
str(inaugTexts)
head(docvars(inaugCorpus), 10)
mycorpus <- corpus(inaugTexts)
```

---

kwic

---

*List key words in context from a text or a corpus of texts.*


---

## Description

For a text or a collection of texts (in a quanteda corpus object), return a list of a keyword supplied by the user in its immediate context, identifying the source text and the word index number within the source text. (Not the line number, since the text may or may not be segmented using end-of-line delimiters.)

## Usage

```
kwic(x, word, window = 5, regex = TRUE)
```

```
## S3 method for class 'character'
```

```
kwic(x, word, window = 5, regex = TRUE)
```

```
## S3 method for class 'corpus'
```

```
kwic(x, word, window = 5, regex = TRUE)
```

## Arguments

x	A text character scalar or a quanteda corpus. (Currently does not support character vectors.)
word	A keyword chosen by the user.
window	The number of context words to be displayed around the keyword.
regex	If TRUE (default), then "word" is a regular expression, otherwise only match the whole word. Note that if regex=TRUE and no special regular expression characters are used in the search query, then the concordance will include all words in which the search term appears, and not just when it appears as an entire word. (For instance, searching for the word "key" will also return "whiskey".)

## Value

A data frame with the context before (preword), the keyword in its original format (word, preserving case and attached punctuation), and the context after (postword). The rows of the dataframe will be named with the word index position, or the text name and the index position for a corpus object.

## Author(s)

Kenneth Benoit and Paul Nulty

## Examples

```
kwic(inaugTexts, "terror")
kwic(inaugTexts, "terror", regex=FALSE) # returns only whole word, without trailing punctuation
```



---

language	<i>get or set the language of corpus documents</i>
----------	--

---

### Description

Get or set the `_language` document-level metadata field in a corpus.

### Usage

```
language(corp)
```

```
language(corp) <- value
```

### Arguments

corp	a corpus object
value	the new value for the language meta-data field, a string or character vector equal in length to <code>ndoc(corp)</code>

### Details

This function modifies the `_language` value set by [metadoc](#). It is a wrapper for `metadoc(corp, "language")`.

---

metacorporus	<i>get or set corpus metadata</i>
--------------	-----------------------------------

---

### Description

Get or set the corpus-level metadata in a quanteda corpus object.

### Usage

```
metacorporus(corp, field = NULL)
```

```
metacorporus(corp, field) <- value
```

### Arguments

corp	A quanteda corpus object
field	Metadata field name(s). If NULL (default), return all metadata names.
value	new value of the corpus metadata field

### Value

For `metacorporus`, a list of the metadata fields in the corpus. If a list is not what you wanted, you can wrap the results in [unlist](#), but this will remove any metadata field that is set to NULL.

For `metacorporus <-`, the corpus with the updated metadata.

**Examples**

```
metacorporus(inaugCorpus)
metacorporus(inaugCorpus, "source")
metacorporus(inaugCorpus, "citation") <- "Presidential Speeches Online Project (2014)."
```

---

metadoc	<i>get or set document-level meta-data</i>
---------	--

---

**Description**

Get or set the document-level meta-data, including reserved fields for language and corpus.

**Usage**

```
metadoc(corp, field = NULL)

metadoc(corp, field = NULL) <- value
```

**Arguments**

corp	A quanteda corpus object
field	string containing the name of the metadata field(s) to be queried or set
value	the new value of the new meta-data field

**Value**

For texts, a character vector of the texts in the corpus.

For texts <-, the corpus with the updated texts.

**Note**

Document-level meta-data names are preceded by an underscore character, such as `_encoding`, but when named in in the `field` argument, do *not* need the underscore character.

**Examples**

```
mycorp <- subset(inaugCorpus, Year>1990)
summary(mycorp, showmeta=TRUE)
metadoc(mycorp, "encoding") <- "UTF-8"
metadoc(mycorp)
metadoc(mycorp, "language") <- "english"
summary(mycorp, showmeta=TRUE)
```

---

ndoc	<i>get the number of documents or features</i>
------	--

---

### Description

Returns the number of documents or features in a quanteda object.

### Usage

```
ndoc(x)

## S3 method for class 'corpus'
ndoc(x)

## S3 method for class 'dfm'
ndoc(x, ...)

nfeature(x)

## S3 method for class 'corpus'
nfeature(x)

## S3 method for class 'dfm'
nfeature(x)
```

### Arguments

x	a corpus or dfm object
...	additional parameters

### Value

an integer (count) of the number of documents or features in the corpus or dfm

### Examples

```
ndoc(inaugCorpus)
ndoc(dfm(inaugCorpus))
nfeature(dfm(inaugCorpus))
nfeature(trimdfm(dfm(inaugCorpus), minDoc=5, minCount=10))
```

---

ngrams	<i>Create ngrams</i>
--------	----------------------

---

### Description

Create a set of ngrams (words in sequence) from text(s) in a character vector

**Usage**

```
ngrams(text, n = 2, concatenator = "_", include.all = FALSE, ...)
```

**Arguments**

text	character vector containing the texts from which ngrams will be extracted
n	the number of tokens to concatenate. Default is 2 for bigrams.
concatenator	character for combining words, default is _ (underscore) character
include.all	if TRUE, add n-1...1 grams to the returned list
...	additional parameters

**Details**

... provides additional arguments passed to [tokenize](#)

**Value**

a list of character vectors of ngrams, one list element per text

**Author(s)**

Ken Benoit, Kohei Watanabe, Paul Nulty

**Examples**

```
ngrams("The quick brown fox jumped over the lazy dog.", n=2)
identical(ngrams("The quick brown fox jumped over the lazy dog.", n=2),
          bigrams("The quick brown fox jumped over the lazy dog.", n=2))
ngrams("The quick brown fox jumped over the lazy dog.", n=3)
ngrams("The quick brown fox jumped over the lazy dog.", n=3, concatenator="~")
ngrams("The quick brown fox jumped over the lazy dog.", n=3, include.all=TRUE)
```

---

nresample

*get the number of resamples*


---

**Description**

Get the number of resamples from a corpus or dfm object

**Usage**

```
nresample(x)

## S3 method for class 'corpus'
nresample(x)

## S3 method for class 'dfm'
nresample(x)
```

**Arguments**

x corpus object containing the texts to be resampled

**Value**

an integer as the number of resampled texts

---

plot.dfm	<i>plot features as a wordcloud</i>
----------	-------------------------------------

---

**Description**

The default plot method for a [dfm](#) object. Produces a wordcloud plot for the features of the dfm, weighted by the total frequencies. To produce word cloud plots for specific documents, the only way currently to do this is to produce a dfm only from the documents whose features you want plotted.

**Usage**

```
## S3 method for class 'dfm'
plot(x, ...)
```

**Arguments**

x a dfm object

... additional parameters passed to [wordcloud](#) or to [text](#) (and [strheight](#), [strwidth](#))

**See Also**

[wordcloud](#)

**Examples**

```
# plot the features (without stopwords) from Obama's two inaugural addresses
mydfm <- dfm(subset(inaugCorpus, President=="Obama"), verbose=FALSE, stopwords=TRUE)
plot(mydfm)

# plot only Lincoln's inaugural address
plot(dfm(subset(inaugCorpus, President=="Lincoln"), verbose=FALSE, stopwords=TRUE))

# plot in colors with some additional options passed to wordcloud
plot(mydfm, random.color=TRUE, rot.per=.25, colors=sample(colors()[2:128], 5))
```

---

print.dfm	<i>print a dfm object</i>
-----------	---------------------------

---

### Description

print method for dfm objects

### Usage

```
## S3 method for class 'dfm'
print(x, show.values = FALSE, show.settings = FALSE, ...)
```

### Arguments

x	the dfm to be printed
show.values	print the dfm as a matrix or array (if resampled).
show.settings	Print the settings used to create the dfm. See <a href="#">settings</a> .
...	further arguments passed to or from other methods

---

quanteda	<i>An R package for the quantitative analysis of textual data.</i>
----------	--

---

### Description

A set of functions for creating and managing text corpora, extracting features from text corpora, and analyzing those features using quantitative methods.

### Author(s)

Ken Benoit and Paul Nulty

---

readLIWCdict	<i>Import a LIWC-formatted dictionary</i>
--------------	---

---

### Description

Make a flattened dictionary list object from a LIWC dictionary file.

### Usage

```
readLIWCdict(path, maxcats = 10, enc = "")
```

**Arguments**

path	full pathname of the LIWC-formatted dictionary file (usually a file ending in .dic)
maxcats	the maximum number of categories to read in, set by the maximum number of dictionary categories that a term could belong to. For non-exclusive schemes such as the LIWC, this can be up to 7. Set to 10 by default, which ought to be more than enough.
enc	a valid input encoding for the file to be read, see <a href="#">iconvlist</a>

**Value**

a dictionary class named list, where each the name of element is a bottom level category in the hierarchical wordstat dictionary. Each element is a list of the dictionary terms corresponding to that level.

**Author(s)**

Kenneth Benoit

**Examples**

```
## Not run:
LIWCdict <- readLIWCdict("~/Dropbox/QUANTESS/corpora/LIWC/LIWC2001_English.dic")
## End(Not run)
```

---

readWStatDict	<i>Import a Wordstat dictionary</i>
---------------	-------------------------------------

---

**Description**

Make a flattened list from a hierarchical wordstat dictionary

**Usage**

```
readWStatDict(path, enc = "", lower = TRUE)
```

**Arguments**

path	full pathname of the wordstat dictionary file (usually ending in .cat)
enc	a valid input encoding for the file to be read, see <a href="#">iconvlist</a>
lower	if TRUE (default), convert the dictionary entries to lower case

**Value**

a named list, where each the name of element is a bottom level category in the hierarchical wordstat dictionary. Each element is a list of the dictionary terms corresponding to that level.

**Author(s)**

Kohei Watanabe, Kenneth Benoit

**Examples**

```
## Not run:
path <- '~/Dropbox/QUANTESS/corpora/LaverGarry.cat'
lgdict <- readWStatDict(path)

## End(Not run)
```

resample

*resampling methods for a corpus***Description**

Draw a set of random resamples from a corpus object, at a specified level of resampling, and record additional "resampled texts" as document-level metadata, stored as `_resampleXXX` for the XXXth resample.

**Usage**

```
resample(x, ...)

## S3 method for class 'corpus'
resample(x, n = 100, unit = c("sentences", "paragraphs"),
  ...)

is.resampled(x)

## S3 method for class 'corpus'
is.resampled(x)

## S3 method for class 'dfm'
is.resampled(x)
```

**Arguments**

<code>x</code>	corpus object containing the texts to be resampled
<code>...</code>	additional arguments passed to <a href="#">segment</a>
<code>n</code>	number of resamples to be drawn
<code>unit</code>	resampling unit for drawing the random samples, can be sentences or paragraphs.

**Details**

`is.resampled` checks a corpus or dfm object and returns TRUE if these contain resampled texts or the results of resampled texts

**Value**

a corpus object containing new resampled texts.

**Note**

Additional resampling units to be added will include fixed length samples and random length samples.



## Examples

```
testCorp <- resample(subset(inaugCorpus, Year>2000), 10, "sentences")
testCorpPara <- resample(corpus(uk2010immig), 10, "paragraphs")
names(metadoc(testCorp))
x <- corpus(c("Sentence One C1. Sentence Two C1. Sentence Three C1.",
              "Sentence One C2. Sentence Two C2. Sentence Three C2. Sentence Four C2. Sentence Five C2. Sentence Six C2.",
              docnames=c("docTwo", "docOne")))
testRS <- resample(x, n=3)
metadoc(testRS)
testRS$documents

# tests to see if a corpus contains resampled texts
is.resampled(testCorp)
is.resampled(inaugCorpus)
```

---

segmentSentence	<i>segment texts into component elements</i>
-----------------	--

---

## Description

Segment text(s) into tokens, sentences, paragraphs, or other sections. `segment` works on a character vector or corpus object, and allows the delimiters to be defined. See details.

## Usage

```
segmentSentence(x, delimiter = "[.!?:;]")

segmentParagraph(x, delimiter = "\\n{2}")

segment(x, ...)

## S3 method for class 'character'
segment(x, what = c("tokens", "sentences", "paragraphs",
                    "other"), delimiter = ifelse(what == "tokens", " ", ifelse(what ==
                    "sentences", "[.!?:;]", "\\n{2}")), ...)

## S3 method for class 'corpus'
segment(x, what = c("tokens", "sentences", "paragraphs",
                    "other"), delimiter = ifelse(what == "tokens", " ", ifelse(what ==
                    "sentences", "[.!?:;]", "\\n{2}")), ...)
```

## Arguments

<code>x</code>	text or corpus object to be segmented
<code>delimiter</code>	delimiter defined as a <a href="#">regex</a> for segmentation. Each type has its own default, except other, which requires a value to be specified.
<code>...</code>	provides additional arguments to be passed to <a href="#">clean</a>
<code>what</code>	unit of segmentation. Current options are tokens, sentences, paragraphs, and other. Segmenting on other allows segmentation of a text on any user-defined value, and must be accompanied by the <code>delimiter</code> argument.

## Details

Tokens are delimited by whitespace. For sentences, the delimiter can be defined by the user. The default for sentences includes ., !, ?, plus ; and :.

For paragraphs, the default is two carriage returns, although this could be changed to a single carriage return by changing the value of `delimiter` to `"\\n{1}"` which is the R version of the [regex](#) for one newline character. (You might need this if the document was created in a word processor, for instance, and the lines were wrapped in the window rather than being hard-wrapped with a newline character.)

## Value

`segmentSentence` returns a character vector of sentences that have been segmented

`segmentParagraph` returns a character vector of paragraphs that have been segmented

A list of segmented texts, with each element of the list corresponding to one of the original texts.

## Examples

```
# segment sentences of the UK 2010 immigration sections of manifestos
segmentSentence(uk2010immig[1])[1:5] # 1st 5 sentences from first (BNP) text
str(segmentSentence(uk2010immig[1])) # a 143-element char vector
str(segmentSentence(uk2010immig[1:2])) # a 155-element char vector (143+ 12)
# segment paragraphs
segmentParagraph(uk2010immig[3])[1:2] # 1st 2 Paragraphs from 3rd (Con) text
str(segmentParagraph(uk2010immig[3])) # a 12-element char vector
# same as tokenize()
identical(tokenize(uk2010immig, lower=FALSE), segment(uk2010immig, lower=FALSE))

# segment into paragraphs
segment(uk2010immig[3:4], "paragraphs")

# segment a text into sentences
segmentedChar <- segment(uk2010immig, "sentences")
segmentedChar[2]
# segment a corpus into sentences
segmentedCorpus <- segment(corpus(uk2010immig), "sentences")
identical(segmentedCorpus, segmentedChar)
```

---

settings

*Get or set the corpus settings*

---

## Description

Get or set the corpus settings

Get or set various settings in the corpus for the treatment of texts, such as rules for stemming, stopwords, collocations, etc.

Get the settings from a which a [dfm](#) was created

**Usage**

```

settings(x, ...)

## S3 method for class 'corpus'
settings(x, field = NULL, ...)

settings(x, field) <- value

## S3 method for class 'dfm'
settings(x, ...)

```

**Arguments**

x	object from/to which settings are queried or applied
...	additional arguments
field	string containing the name of the setting to be set or queried settings(x) query the corps settings settings(x, field) <- update the corpus settings for field
value	new setting value

**Examples**

```

settings(inaugCorpus, "stopwords")
tempdfm <- dfm(inaugCorpus)
tempdfmSW <- dfm(inaugCorpus, stopwords=TRUE)
settings(inaugCorpus, "stopwords") <- TRUE
tempdfmSW <- dfm(inaugCorpus)
tempdfm <- dfm(inaugCorpus, stem=TRUE)
settings(tempdfm)

```

---

settingsInitialize	<i>settingsInitialize returns a list of legal settings, set to their default values</i>
--------------------	---

---

**Description**

settingsInitialize returns a list of legal settings, set to their default values

**Usage**

```
settingsInitialize()
```

similarity

*compute similarities between documents and/or features***Description**

Compute similarities between documents and/or features from a [dfm](#). Uses the similarity measures defined in [simil](#). See [pr\\_DB](#) for available distance measures, or how to create your own.

**Usage**

```
similarity(x, selection, n = 10, margin = c("features", "documents"),
  method = "correlation", sort = TRUE, normalize = TRUE, digits = 4)
```

**Arguments**

x	a <a href="#">dfm</a> object
selection	character or character vector of document names or feature labels from the dfm
n	the top n most similar items will be returned, sorted in descending order. If n is NULL, return all items.
margin	identifies the margin of the dfm on which similarity will be computed: features for word/term features or documents for documents.
method	a valid method for computing similarity from <a href="#">pr_DB</a>
sort	sort results in descending order if TRUE
normalize	if TRUE, normalize the dfm by term frequency within document (so that the dfm values will be relative term frequency within each document)
digits	digits for rounding results

**Value**

a named list of the selection labels, with a sorted named vector of similarity measures.

**Note**

The method for computing feature similarities can be quite slow when there are large numbers of feature types. Future implementations will hopefully speed this up.

**Examples**

```
# create a dfm from inaugural addresses from Reagan onwards
presDfm <- dfm(subset(inaugCorpus, Year>1980), stopwords=TRUE, stem=TRUE)

# compute some document similarities
similarity(presDfm, "1985-Reagan", n=5, margin="documents")
similarity(presDfm, c("2009-Obama" , "2013-Obama"), n=5, margin="documents")
similarity(presDfm, c("2009-Obama" , "2013-Obama"), n=NULL, margin="documents")
similarity(presDfm, c("2009-Obama" , "2013-Obama"), n=NULL, margin="documents", method="cosine")
similarity(presDfm, "2005-Bush", n=NULL, margin="documents", method="eJaccard", sort=FALSE)

## Not run:
# compute some term similarities
```

```

similarity(presDfm, c("fair", "health", "terror"), method="cosine")

# compare to tm
require(tm)
data("crude")
crude <- tm_map(crude, content_transformer(tolower))
crude <- tm_map(crude, removePunctuation)
crude <- tm_map(crude, removeNumbers)
crude <- tm_map(crude, stemDocument)
tdm <- TermDocumentMatrix(crude)
findAssocs(tdm, c("oil", "opec", "xyz"), c(0.75, 0.82, 0.1))
# in quanteda
crudeDfm <- dfm(corpus(crude))
similarity(crudeDfm, c("oil", "opec", "xyz"), normalize=FALSE, digits=2)

## End(Not run)

```

---

sort.dfm

*sort a dfm by one or more margins*


---

## Description

Sorts a [dfm](#) by frequency of total features, total features in documents, or both

## Usage

```

## S3 method for class 'dfm'
sort(x, decreasing = TRUE, margin = c("features", "docs",
  "both"), ...)

```

## Arguments

x	Document-feature matrix created by <a href="#">dfm</a>
decreasing	TRUE (default) if sort will be in descending order, otherwise sort in increasing order
margin	which margin to sort on features to sort by frequency of features, docs to sort by total feature counts in documents, and both to sort by both
...	additional argumnets passed to base method <code>sort.int</code>

## Value

A sorted [dfm](#) matrix object

## Author(s)

Ken Benoit

## Examples

```
dtm <- dfm(inaugCorpus)
dtm[1:10, 1:5]
dtm <- sort(dtm)
sort(dtm)[1:10, 1:5]
sort(dtm, TRUE, "both")[1:10, 1:5] # note that the decreasing=TRUE argument
                                   # must be second, because of the order of the
                                   # formals in the generic method of sort()
```

---

statLexdiv	<i>calculate lexical diversity</i>
------------	------------------------------------

---

## Description

Calculate the lexical diversity or complexity of text(s).

## Usage

```
statLexdiv(x, ...)

## S3 method for class 'dfm'
statLexdiv(x, measure = c("TTR", "C", "R", "CTTR", "U", "S",
  "Maas"), log.base = 10, ...)

## S3 method for class 'numeric'
statLexdiv(x, measure = c("TTR", "C", "R", "CTTR", "U", "S",
  "Maas"), log.base = 10, ...)
```

## Arguments

x	a <a href="#">document-feature matrix object</a>
...	additional arguments
measure	A character vector defining the measure to calculate.
log.base	A numeric value defining the base of the logarithm.

## Details

statLexdiv calculates a variety of proposed indices for lexical diversity. In the following formulae,  $N$  refers to the total number of tokens, and  $V$  to the number of types:

"TTR": The ordinary *Type-Token Ratio*:

$$TTR = \frac{V}{N}$$

"C": Herdan's *C* (Herdan, 1960, as cited in Tweedie & Baayen, 1998; sometimes referred to as *LogTTR*):

$$C = \frac{\lg V}{\lg N}$$

"R": Guiraud's *Root TTR* (Guiraud, 1954, as cited in Tweedie & Baayen, 1998):

$$R = \frac{V}{\sqrt{N}}$$

"CTTR": Carroll's *Corrected TTR*:

$$CTTR = \frac{V}{\sqrt{2N}}$$

"U": Dugast's *Uber Index* (Dugast, 1978, as cited in Tweedie & Baayen, 1998):

$$U = \frac{(\lg N)^2}{\lg N - \lg V}$$

"S": Summer's index:

$$S = \frac{\lg \lg V}{\lg \lg N}$$

"K": Yule's *K* (Yule, 1944, as cited in Tweedie & Baayen, 1998) is calculated by:

$$K = 10^4 \times \frac{(\sum_{X=1}^X f_X X^2) - N}{N^2}$$

where  $N$  is the number of tokens,  $X$  is a vector with the frequencies of each type, and  $f_X$  is the frequencies for each  $X$ .

"Maas": Maas' indices ( $a$ ,  $\lg V_0$  &  $\lg_e V_0$ ):

$$a^2 = \frac{\lg N - \lg V}{\lg N^2}$$

$$\lg V_0 = \frac{\lg V}{\sqrt{1 - \frac{\lg V^2}{\lg N^2}}}$$

The measure was derived from a formula by M"uller (1969, as cited in Maas, 1972).  $\lg_e V_0$  is equivalent to  $\lg V_0$ , only with  $e$  as the base for the logarithms. Also calculated are  $a$ ,  $\lg V_0$  (both not the same as before) and  $V'$  as measures of relative vocabulary growth while the text progresses. To calculate these measures, the first half of the text and the full text will be examined (see Maas, 1972, p. 67 ff. for details). Note: for the current method (for a dfm) there is no computation on separate halves of the text.

## Value

a vector of lexical diversity statistics, each corresponding to an input document

## Note

This implements only the static measures of lexical diversity, not more complex measures based on windows of text such as the Mean Segmental Type-Token Ratio, the Moving-Average Type-Token Ratio (Covington & McFall, 2010), the MLTD or MLTD-MA (Moving-Average Measure of Textual Lexical Diversity) proposed by McCarthy & Jarvis (2010) or Jarvis (no year), or the HD-D version of vocd-D (see McCarthy & Jarvis, 2007). These are available from the package **korRpus**.

## Author(s)

Kenneth Benoit, adapted from the S4 class implementation written by Meik Michalke in the **korRpus** package.

## References

- Covington, M.A. & McFall, J.D. (2010). Cutting the Gordian Knot: The Moving-Average Type-Token Ratio (MATTR). *Journal of Quantitative Linguistics*, 17(2), 94–100.
- Maas, H.-D., (1972). \ "Uber den Zusammenhang zwischen Wortschatzumfang und L"ange eines Textes. *Zeitschrift f"ur Literaturwissenschaft und Linguistik*, 2(8), 73–96.
- McCarthy, P.M. & Jarvis, S. (2007). vocd: A theoretical and empirical evaluation. *Language Testing*, 24(4), 459–488.
- McCarthy, P.M. & Jarvis, S. (2010). MTLT, vocd-D, and HD-D: A validation study of sophisticated approaches to lexical diversity assessment. *Behaviour Research Methods*, 42(2), 381–392.
- Michalke, Meik. (2014) *koRpus: An R Package for Text Analysis*. Version 0.05-5. <http://reaktanz.de/?c=hacking&s=koRpus>
- Tweedie, F.J. & Baayen, R.H. (1998). How Variable May a Constant Be? Measures of Lexical Richness in Perspective. *Computers and the Humanities*, 32(5), 323–352.

## Examples

```
mydfm <- dfm(inaugCorpus)
mydfmSW <- dfm(inaugCorpus, stopwords=TRUE)
results <- data.frame(TTR = statLexdiv(mydfm, "TTR"),
                      CTTR = statLexdiv(mydfm, "CTTR"),
                      U = statLexdiv(mydfm, "U"),
                      TTRs = statLexdiv(mydfmSW, "TTR"),
                      CTTRs = statLexdiv(mydfmSW, "CTTR"),
                      Us = statLexdiv(mydfmSW, "U"))

results
cor(results)
t(statLexdiv(mydfmSW, "Maas"))
```

---

stopwords

*A named list containing common stopwords in 14 languages*

---

## Description

SMART English stopwords from the SMART information retrieval system (obtained from <http://jmlr.csail.mit.edu/papers/smart-stop-list/english.stop>) and a set of stopword lists from the Snowball stemmer project in different languages (obtained from [http://svn.tartarus.org/snowball/trunk/website/algorithms/\\*/stop.txt](http://svn.tartarus.org/snowball/trunk/website/algorithms/*/stop.txt)). Supported languages are danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, portuguese, russian, spanish, and swedish. Language names are case sensitive. Alternatively, their IETF language tags may be used.



---

stopwordsGet	<i>access stopwords</i>
--------------	-------------------------

---

### Description

This function retrieves stopwords from the type specified in the `kind` argument and returns the stopword list as a character vector. The default is English. See [stopwords](#) for information about the list.

### Usage

```
stopwordsGet(kind = "english")
```

### Arguments

<code>kind</code>	The pre-set kind of stopwords (as a character string)
-------------------	---

### Value

a character vector or dfm with stopwords removed

### Examples

```
stopwordsGet()
stopwordsGet("italian")
```

---

stopwordsRemove	<i>remove stopwords from a text or dfm</i>
-----------------	--

---

### Description

This function takes a character vector or [dfm](#) and removes words in the remove common or 'semantically empty' words from a text. See [stopwordsGet](#) for the information about the default lists.

### Usage

```
stopwordsRemove(text, stopwords = NULL)

## S3 method for class 'character'
stopwordsRemove(text, stopwords = NULL)

## S3 method for class 'dfm'
stopwordsRemove(text, stopwords = NULL)
```

### Arguments

<code>text</code>	Text from which stopwords will be removed
<code>stopwords</code>	Character vector of stopwords to remove - if none is supplied, a default set of English stopwords is used

## Details

This function takes a character vector 'text' and removes words in the list provided in stopwords. If no list of stopwords is provided a default list for English is used. The function [stopwordsGet](#) can load a default set of stopwords for many languages.

## Value

a character vector or dfm with stopwords removed

## Examples

```
## examples for character objects
someText <- "Here is an example of text containing some stopwords we want to remove."
itText <- "Ecco un esempio di testo contenente alcune parole non significative che vogliamo rimuovere."
stopwordsRemove(someText)
stopwordsRemove(someText, stopwordsGet("SMART"))
stopwordsRemove(itText, stopwordsGet("italian"))
stopwordsRemove(someText, c("containing", "example"))

## example for dfm objects
docmat <- dfm(uk2010immig)
docmatNostopwords <- stopwordsRemove(docmat)
dim(docmat)
dim(docmatNostopwords)
dim(stopwordsRemove(docmat, stopwordsGet("SMART")))
```

---

subset.corpus	<i>extract a subset of a corpus</i>
---------------	-------------------------------------

---

## Description

Works just like the normal subset command but for corpus objects

## Usage

```
## S3 method for class 'corpus'
subset(x, subset = NULL, select = NULL, ...)
```

## Arguments

x	corpus object to be subsetted.
subset	logical expression indicating elements or rows to keep: missing values are taken as false.
select	expression, indicating the attributes to select from the corpus
...	additional arguments affecting the summary produced

## Value

corpus object

**Examples**

```
summary(subset(inaugCorpus, Year>1980))
summary(subset(inaugCorpus, Year>1930 & President=="Roosevelt", select=Year))
```

---

summary.corpus	<i>Corpus summary</i>
----------------	-----------------------

---

**Description**

Displays information about a corpus object, including attributes and metadata such as date of number of texts, creation and source.

**Usage**

```
## S3 method for class 'corpus'
summary(object, n = 100, verbose = TRUE,
        showmeta = FALSE, ...)
```

**Arguments**

object	corpus to be summarized
n	maximum number of texts to describe, default=100
verbose	FALSE to turn off printed output
showmeta	TRUE to include document-level meta-data
...	additional arguments affecting the summary produced

**Examples**

```
summary(inaugCorpus)
summary(inaugCorpus, n=10)
mycorpus <- corpus(uk2010immig, docvars=data.frame(party=names(uk2010immig)), enc="UTF-8")
summary(mycorpus, showmeta=TRUE) # show the meta-data
mysummary <- summary(mycorpus, verbose=FALSE) # (quietly) assign the results
mysummary$Types / mysummary$Tokens           # crude type-token ratio
```

---

syllableCounts	<i>A named list mapping words to counts of their syllables</i>
----------------	--

---

**Description**

A named list mapping words to counts of their syllables, generated from the CMU pronunciation dictionary

**References**

<http://www.speech.cs.cmu.edu/cgi-bin/cmudict>

**Examples**

```
data(syllableCounts)
syllableCounts["sixths"]
syllableCounts["onomatopoeia"]
```

texts	<i>get or set corpus texts</i>
-------	--------------------------------

---

**Description**

Get or replace the texts in a quanteda corpus object.

**Usage**

```
texts(corp)

texts(corp) <- value
```

**Arguments**

corp	A quanteda corpus object
value	character vector of the new texts

**Value**

For texts, a character vector of the texts in the corpus.  
For texts <-, the corpus with the updated texts.

**Examples**

```
texts(inaugCorpus)[1]
sapply(texts(inaugCorpus), nchar) # length in characters of the inaugural corpus texts

## this doesn't work yet - need to overload `[` for this replacement function
# texts(inaugTexts)[55] <- "GW Bush's second inaugural address, the condensed version."
```

---

tf	<i>normalizes the term frequencies a dfm</i>
----	--

---

**Description**

Returns a matrix of term weights, as a [dfm](#) object

**Usage**

```
tf(x)
```

**Arguments**

x	Document-feature matrix created by <a href="#">dfm</a>
---	--

**Value**

A dfm matrix object where values are relative term proportions within the document

**Author(s)**

Ken Benoit

**Examples**

```
data(inaugCorpus)
dtm <- dfm(inaugCorpus)
dtm[1:10, 100:110]
tf(dtm)[1:10, 100:110]
```

---

tfidf	<i>compute the tf-idf weights of a dfm</i>
-------	--

---

**Description**

Returns a matrix of tf-idf weights, as a [dfm](#) object

**Usage**

```
tfidf(x, normalize = TRUE)

## S3 method for class 'dfm'
tfidf(x, normalize = TRUE)
```

**Arguments**

x	document-feature matrix created by <a href="#">dfm</a>
normalize	whether to normalize term frequency by document totals

**Value**

A dfm matrix object where values are tf-idf weights

**Author(s)**

Ken Benoit

**Examples**

```
data(inaugCorpus)
dtm <- dfm(inaugCorpus)
dtm[1:10, 100:110]
tfidf(dtm)[1:10, 100:110]
tfidf(dtm, normalize=FALSE)[1:10, 100:110]
```

---

tokenize	<i>tokenize a set of texts</i>
----------	--------------------------------

---

## Description

Tokenize the texts from a character vector or from a corpus.

## Usage

```
tokenize(x, ...)

## S3 method for class 'character'
tokenize(x, simplify = FALSE, sep = " ", ...)

## S3 method for class 'corpus'
tokenize(x, ...)
```

## Arguments

x	The text(s) or corpus to be tokenized
...	additional arguments passed to <a href="#">clean</a>
simplify	If TRUE, return a character vector of tokens rather than a list of length <a href="#">ndoc</a> (texts), with each element of the list containing a character vector of the tokens corresponding to that text.
sep	by default, tokenize expects a "white-space" delimiter between tokens. Alternatively, sep can be used to specify another character which delimits fields.

## Value

A list of length [ndoc](#)(x) of the tokens found in each text.

A list of length [ndoc](#)(texts) of the tokens found in each text.

## Examples

```
# same for character vectors and for lists
tokensFromChar <- tokenize(inaugTexts)
tokensFromCorp <- tokenize(inaugCorpus)
identical(tokensFromChar, tokensFromCorp)
str(tokensFromChar)
# returned as a list
head(tokenize(inaugTexts[57])[[1]], 10)
# returned as a character vector using simplify=TRUE
head(tokenize(inaugTexts[57], simplify=TRUE), 10)

# demonstrate some options with clean
head(tokenize(inaugTexts[57], simplify=TRUE, lower=FALSE), 30)
```

---

topfeatures	<i>list the most frequent features</i>
-------------	--

---

### Description

List the most frequently occurring features in a [dfm](#)

### Usage

```
topfeatures(x, n = 10, decreasing = TRUE, ci = 0.95)
```

```
## S3 method for class 'dfm'
```

```
topfeatures(x, n = 10, decreasing = TRUE, ci = 0.95)
```

### Arguments

x	the object whose features will be returned
n	how many top features should be returned
decreasing	If TRUE, return the n most frequent features, if FALSE, return the n least frequent features
ci	confidence interval from 0-1.0 for use if dfm is resampled

### Value

A named numeric vector of feature counts, where the names are the feature labels.

### Examples

```
topfeatures(dfm(inaugCorpus))
topfeatures(dfm(inaugCorpus, stopwords=TRUE))
# least frequent features
topfeatures(dfm(inaugCorpus), decreasing=FALSE)
```

---

trimdfm	<i>Trim a dfm based on a subset of features and words</i>
---------	---

---

### Description

Returns a document by feature matrix reduced in size based on document and term frequency, and/or subsampling.

### Usage

```
trimdfm(x, minCount = 1, minDoc = 1, sample = NULL, keep = NULL,
        verbose = TRUE)
```

Arguments

x	document-feature matrix created by <a href="#">dfm</a>
minCount	minimum feature count
minDoc	minimum number of documents in which a feature appears
sample	how many features to retain (based on random selection)
keep	regular expression specifying which features to keep
verbose	print messages

Value

A [dfm](#) object reduced in size.

Author(s)

Ken Benoit adapted from code originally by Will Lowe (see [trim](#))

Examples

```
dtm <- dfm(inaugCorpus)
dim(dtm)
dtmReduced <- trimdfm(dtm, minCount=10, minDoc=2) # only words occuring at least 5 times and in at least 2 documents
dim(dtmReduced)
dtmReduced <- trimdfm(dtm, keep="^nation|^citizen|^union$")
topfeatures(dtmReduced, NULL)
dtmSampled <- trimdfm(dtm, sample=200) # top 200 words
dim(dtmSampled) # 196 x 200 words
```

---

uk2010immig	<i>Immigration-related sections of 2010 UK party manifestos</i>
-------------	---

---

Description

Extracts from the election manifestos of 9 UK political parties from 2010, related to immigration or asylum-seekers.

Format

A named character vector of plain ASCII texts

Examples

```
data(uk2010immig)
uk2010immigCorpus <- corpus(uk2010immig, docvars=list(party=names(uk2010immig)))
language(uk2010immigCorpus) <- "english"
encoding(uk2010immigCorpus) <- "UTF-8"
summary(uk2010immigCorpus)
```



---

wordstem	<i>stem words</i>
----------	-------------------

---

### Description

Apply a stemmer to words. This is a wrapper to [wordStem](#) designed to allow this function to be called without loading the entire **SnowballC** package. [wordStem](#) uses Dr. Martin Porter's stemming algorithm and the C libstemmer library generated by Snowball.

### Usage

```
wordstem(words, language = "porter")
```

### Arguments

words	a character vector of words whose stems are to be extracted.
language	the name of a recognized language, as returned by <code>getStemLanguages</code> , or a two- or three-letter ISO-639 code corresponding to one of these languages (see references for the list of codes)

### Value

A character vector with as many elements as there are in the input vector with the corresponding elements being the stem of the word. Elements of the vector are converted to UTF-8 encoding before the stemming is performed, and the returned elements are marked as such when they contain non-ASCII characters.

### See Also

[wordStem](#); <http://snowball.tartarus.org/>.

### Examples

```
# Simple example
wordstem(c("win", "winning", "winner"))
```

---

zipfiles	<i>unzip a zipped collection of text files and return the directory</i>
----------	---

---

### Description

zipfiles extracts a set of text files in a zip archives, and returns the name of the temporary directory where they are stored. It can be passed to [corpus.directory](#) for import.

### Usage

```
zipfiles(zfile = NULL, ...)
```

**Arguments**

zfile	a character string specifying the name (including path) of the zipped file, or a URL naming the file (see example); or NULL to use a GUI to choose a file from disk
...	additional arguments passed to <a href="#">unzip</a>

**Value**

a [directory](#) class object containing the unzipped files

**Examples**

```
## Not run:
# from a zip file on the web
myzipcorp <- corpus(zipfiles("http://kenbenoit.net/files/EUcoalsubsidies.zip"),
  notes="From some EP debate about coal mine subsidies")
docvars(myzipcorp, "speakername") <- docnames(myzipcorp)
summary(myzipcorp)

# call up interactive user input
myzipcorp <- corpus(zipfiles())

## End(Not run)
```

# Index

`+.corpus (corpus)`, 8

`as.dfm (dfm)`, 12

`as.DocumentTermMatrix`, 15

`attributes`, 13

`bigrams`, 3

`changeunits`, 4

`clean`, 4, 12, 33, 46

`collocations`, 5

`compoundWords`, 7

`corpus`, 8, 8

`corpus.directory`, 49

`countSyllables`, 11

`data.frame`, 9

`describeTexts`, 11

`dfm`, 9, 12, 12, 13–16, 18, 29, 34, 36, 37, 41, 44, 45, 47, 48

`dfm2ldaformat`, 14

`dfm2tmformat`, 15

`directory`, 8, 15, 50

`docnames`, 16

`docnames<- (docnames)`, 16

`document-feature matrix object`, 38

`DocumentTermMatrix`, 15

`docvars`, 9, 10, 17

`docvars<- (docvars)`, 17

`Encoding`, 18, 20, 22

`encoding`, 10, 18

`encoding<- (encoding)`, 18

`features (features.dfm)`, 18

`features.dfm`, 18

`file`, 15

`flatten.dictionary`, 19

`getRootFileNames`, 20

`getTextDir`, 20

`getTextDirGui`, 21

`getTextFiles`, 22

`getTweets`, 22

`iconv`, 9

`iconvlist`, 9, 18, 31

`inaugCorpus`, 23

`inaugTexts (inaugCorpus)`, 23

`is.corpus (corpus)`, 8

`is.dfm (dfm)`, 12

`is.resampled (resample)`, 32

`kwic`, 24

`language`, 10, 25

`language<- (language)`, 25

`lda.collapsed.gibbs.sampler`, 14

`metacorporus`, 10, 25

`metacorporus<- (metacorporus)`, 25

`metadoc`, 9, 10, 18, 25, 26

`metadoc<- (metadoc)`, 26

`ndoc`, 27, 46

`nfeature (ndoc)`, 27

`ngrams`, 27

`nresample`, 28

`plot.dfm`, 29

`pr_DB`, 36

`print.dfm`, 30

`quanteda`, 23, 30

`quanteda-package (quanteda)`, 30

`readLIWCdict`, 30

`readWStatDict`, 31

`regex`, 33, 34

`regular expression`, 5, 9, 20

`resample`, 13, 32

`segment`, 4, 32

`segment (segmentSentence)`, 33

`segmentParagraph (segmentSentence)`, 33

`segmentSentence`, 33

`settings`, 9, 10, 13, 30, 34

`settings<- (settings)`, 34

`settingsInitialize`, 35

`simil`, 36

similarity, [36](#)  
simple triplet matrix, [15](#)  
sort.dfm, [37](#)  
statLexdiv, [38](#)  
stopwords, [12](#), [40](#), [41](#)  
stopwordsGet, [41](#), [41](#), [42](#)  
stopwordsRemove, [41](#)  
strheight, [29](#)  
strwidth, [29](#)  
subset.corpus, [42](#)  
summary.corpus, [43](#)  
syllableCounts, [43](#)  
  
text, [29](#)  
texts, [9](#), [10](#), [44](#)  
texts<- (texts), [44](#)  
tf, [44](#)  
tfidf, [45](#)  
tokenise (tokenize), [46](#)  
tokenize, [3](#), [28](#), [46](#)  
topFeatures (topfeatures), [47](#)  
topfeatures, [47](#)  
trim, [48](#)  
trimdfm, [47](#)  
  
uk2010immig, [48](#)  
unlist, [25](#)  
unzip, [50](#)  
  
VCorpus, [8](#), [10](#)  
  
wordcloud, [29](#)  
wordStem, [49](#)  
wordstem, [49](#)  
  
zipfiles, [8](#), [49](#)